

Variables

CSE 142

Memory

- All the data created and manipulated by a program is stored in the computer's memory
- It has three attributes:
 - Location
 - Value
 - Type

Location	Value	Type	Meaning?
3423	65	char	'A'

Variables

- If we had to name memory locations explicitly by number we'd go crazy, so...
- In our programs we name memory symbolically, by defining variables.
- Each variable must be declared.
- The declaration indicates the name and the type of the variable.
 - The type indicates how to interpret the value stored there.

Declaring Variables

int months;

Integer variables represent whole numbers:

1, 17, -32, 0

Not 1.5, 2.0

float pi;

Floating point variables represent real numbers:

3.14, -27.5, 6.02e23, 5.0

Not 3

char first_initial, middle_initial, marital_status;

Character variables represent individual keyboard characters:

'a', 'b', 'M', '0', '9', '#', ' ' **Not "Bill"**

Memory, Variables, Values, and Types

Location	Value	Type	Possible variable name
1000:	'm'	<i>char</i>	<i>first_initial</i>
1002:	3.1416	<i>float</i>	<i>almost_pi</i>
1016:	-112	<i>int</i>	<i>my_balance</i>
1020:	214	<i>int</i>	<i>your_balance</i>
1024:	'y'	<i>char</i>	<i>answer</i>
1026:	3	<i>int</i>	<i>foobar</i>

Identifiers

- "Identifiers" are names for things in a program
 - for examples, names of variables
- In C, identifiers follow certain rules:
 - use letters, numerals, and underscore (_)
 - do not begin with a numeral
 - cannot be reserved words
 - are "case-sensitive"
 - can be arbitrarily long but...
- *Style point: Good choices for identifiers can be extremely helpful in understanding programs*

Reserved words

- Certain words have a "reserved" (permanent, special) meaning in C
 - We've seen *int*, *float*, *char* already
 - Will see a couple of dozen more eventually
- These words always have that special meaning, and cannot be used for other purposes.
 - Cannot be used names of variables
 - Must be spelled correctly
 - Sometimes also called “keywords”

Declarations vs Statements

- Programs are made up of “declarations” and “statements”
- **Declarations** define something’s NAME and TYPE
 - We’ve already seen variable declarations
 - Later we’ll see constant declarations and function declarations
- **Statements** tell the CPU to do something
 - We’re about to see our first kind of statement
 - Eventually we’ll see about a dozen other kinds of statements

Storing values in variables

- A variable declaration gives a name to a memory location, but... *how do we place a value in that memory location?*
- Placing a value in a location is called "storing"
- One way to store a value is with the **assignment statement**.
 - Later we'll see that *scanf* can also store values into a variable.

Assigning Values

- An assignment statement places a value into a variable.
- The assignment may specify a simple value to be stored, or an expression

```
int area, length, width; /* declaration of 3 variables */
length = 16;           /* length gets 16 */
width = 32;           /* width gets 32 */
area = length * width; /* area gets length times width */
```

- *The result: store the **value** of the expression on the right into the **variable** on the left.*

Types

- C is a strongly typed language
- That means that if the types on the left and right hand sides of an assignment don't match, the compiler will complain.

The Basic Types

char	character	0 to 255
int	integer	-32767 to 32767
float	single precision floating point	six digits of precision
double	double precision floating point	ten digits of precision

- Chars and ints are almost the same thing

Signed & unsigned

- chars and ints can also be adorned with signed & unsigned
- unsigned ints can be a little bigger
- floats & doubles are always signed

int	integer	-32767 to 32767
unsigned int	can't be less than zero	0 to 65535
signed int	can be negative	-32767 to 32767

Long Types

- ANSI C compilers also let you declare an int, float or double as “long” meaning it can hold numbers twice as big

int	-32767 to 32767
long int	-2147483647 to 2147483647
double	Ten digits of precision
long double	Ten digits of precision

Using Types In Code

- Every expression has a type
- Types have to match, or the compiler complains.

```
unsigned int arms = 2;  
int gross = 144;  
float money = 2.99;  
long int distance = 93000000;  
char section = 'b';
```

Using Types In Code

```
/* all these examples are wrong */  
  
unsigned int legs = -1;  
float temperature = 60;    // implicit  
    conversion  
int pi = 3.14159;  
char initial = 4;         // implicit  
    conversion  
double seat_number = 'g';
```

C Program Structure

```
#include <stdio.h>  
int main(void)  
{
```

variable declarations

program statements

```
return(0);  
}
```

The shaded parts will be in *every* program.

C Program Structure

```
#include <stdio.h>  
int main(void)  
{
```

variable declarations

program statements

```
return(0);  
}
```

Notice ALL declarations are before ANY statements (in C).

Problem Solving and Program Design (Review)

- Clearly **specify** the problem
- **Analyze** the problem
- Design an **algorithm** to solve the problem
- **Implement** the algorithm (write the program)
- **Test** and verify the completed program
 - The test-debug cycle
- **Maintain** and update the program

Example Problem: Fahrenheit to Celsius

Problem (specified):

Convert Fahrenheit temperature to Celsius

Algorithm (result of analysis):

$\text{Celsius} = 5/9 (\text{Fahrenheit} - 32)$

What kind of data (result of analysis):

float fahrenheit, celsius;

Fahrenheit to Celsius (I)

An actual C program

```
#include <stdio.h>  
int main(void)  
{  
    float fahrenheit, celsius;  
  
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;  
  
    return(0);  
}
```

Fahrenheit to Celsius (II)

```
#include <stdio.h>
```

```
int main(void)
```

```
{  
    float fahrenheit, celsius;  
    printf("Enter a Fahrenheit temperature: ");  
    scanf("%lf", &fahrenheit);  
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;  
    printf("That equals %f degrees Celsius.\n",  
        celsius);  
    return(0);  
}
```

Running the Program

*Enter a Fahrenheit temperature: 45.5
That equals 7.500000 degrees Celsius*

Program “trace:”

	<u><i>fahrenheit</i></u>	<u><i>celsius</i></u>
after declaration	?	?
after first <i>printf</i>	?	?
after <i>scanf</i>	45.5	?
after assignment	45.5	7.5
after second <i>printf</i>	45.5	7.5

Fahrenheit to Celsius (III)

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
float fahrenheit, celsius;
```

```
printf("Enter a Fahrenheit temperature: ");
```

```
scanf("%lf", &fahrenheit);
```

```
celsius = fahrenheit - 32.0 ;
```

```
celsius = celsius * 5.0 / 9.0 ;
```

```
printf("That equals %f degrees Celsius.\n",  
celsius);
```

```
return(0);
```

```
}
```

Assignment step-by-step

*celsius = celsius * 5.0 / 9.0 ;*

1. Evaluate right-hand side
 - a. Find current value of **celsius** 13.5
 - b. Multiply by 5.0 67.5
 - c. Divide by 9.0 7.5
2. Assign 7.5 to be the new value of **celsius**
(old value 13.5 of celsius is lost.)

$$my_age = my_age + 1$$

- The same variable may appear on both sides of an assignment statement!

```
my_age = my_age + 1 ;  
balance = balance + deposit ;
```

- The **old** value of the variable is used to compute the value of the expression, before the variable is changed.
- *You wouldn't do this in math!*

Note on lecture examples

- Slides often leave out important details

my_age = my_age + 1;

- This is a legal C statement **only if**:
 - *my_age* has previously been declared in the program
 - *my_age* has the proper type (e.g. *int*)
 - the statement occurs in a legal position;
 - the full program has “*int main (void)*”, etc., etc.
- Use your creative powers and common sense to deduce what’s missing in the examples!

Initializing variables

- **Initialization** means giving something a value for the **first** time.
- Anything which changes the value of a variable is a potential way of initializing it.
 - For now, that means assignment statement
- *General rule: variables have to be initialized before their value is used.*
 - Failure to initialize is a common source of bugs.
- Variables in a C program are **not** automatically initialized to 0!

Declaring vs Initializing

```
int main (void) {  
    double income; /* declaration of income,  
                    not an assignment,  
                    not an initialization*/  
    income = 35500.00; /* assignment to income,  
                    initialization of income,  
                    not a declaration.*/  
    printf ("Old income is %f\n", income);  
    income = 39000.00; /* assignment to income,  
                    not a declaration,  
                    not an initialization */  
    printf ("After raise: %f\n", income);  
}
```

Constants

- Sometimes it doesn't make sense to vary
 - Doing it in the preprocessor:
 - `#define PI = 3.14`
 - Doing it via the compiler **C++**
 - `const double PI = 3.14;`
- We prefer the compiler!

Does Terminology Matter?

- "variable", "reserved word", "initialization", "declaration", "assignment", etc., etc.
- You can write a complicated program without using these words
- But you can't talk about your programs without them!
- Learn the exact terminology as you go, and get in the habit of using it.
 - Your instructor, peers, and clients will bless you...
 - ... and will be able to better help you

Homework 1

- Convert the Fahrenheit to Celsius program to convert miles to inches.
- Hint : for integers, change `%lf` in `scanf` to `%d`, and `%f` in `printf` to `%d`.
- Compile and run the program